```
import os, pathlib, shutil, random
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
# Download and extract IMDB dataset
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  16.4M      0  0:00:04  0:00:04 --:--:-- 18.9M
```

```
import os

def summarize_imdb_files(base_path="aclImdb", num_files=5):
    for split in ["train", "test"]:
        print(f"\nSummary of '{split}' split:")
        for sentiment in ["pos", "neg"]:
            print(f"  Sentiment: {sentiment}")
            folder_path = os.path.join(base_path, split, sentiment)
            files = os.listdir(folder_path)[:num_files]  # Read first `num_files` files
            for i, file_name in enumerate(files):
                file_path = os.path.join(folder_path, file_name)
                with open(file_path, "r", encoding="utf-8") as f:
                    content = f.readlines()
                    print(f"\n    File {i + 1}: {file_name}")
                    print(f"    Lines in file: {len(content)}")
                    print("    First 5 lines (or fewer):")
                    print("    " + "".join(content[:5]).strip())

# Run the summary function
summarize_imdb_files()
```

```
    Lines in file: 1
    First 5 lines (or fewer):
    Office work, especially in this era of computers, multi-functional copy machines, e-mail, voice mail, snail mail and `temps,' is territory ripe with satirical possibilities, a vein previously tapped in such films as `Clockwatchers' and `Office Space,' and very successfully. This latest addition to the temp/humor pool, however, `Haiku Tunnel,' directed by Josh Kornbluth and Jacob Kornbluth, fails to live up to it's predecessors, and leaves the laughs somewhere outside the door, waiting for a chance to sneak in. Unfortunately for the audience, that chance never co

Summary of 'test' split:
  Sentiment: pos

    File 1: 5447_7.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Leno talks through his punch lines. This spoils the joke, for those of you who haven't figured it out. His show is held in a micro sized studio for a reason, or two reasons -- the small amount of laughter will be amplified, and few want to see him. Letterman's set is the polar opposite -- Ed Sullivan would be proud and the balcony is always full.<br /><br />NBC, tacky NBC, will never get it. Founder David Sarnoff's megalomaniacal enterprise continues to be all about the money. At least CBS has a bit of a family feel to it.<br /><br />Leno collects cars, or is it

    File 2: 6596_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    When the Bicentennial hit, I was in Hershey, PA and part of the Middletown Skate Team. We didn't do anything except skate around trying to learn what we were reading in the pages of Skateboarder Magazine. A couple of local hills provided all the speed-wobbles you could handle. At the time Mike Weed was the hot stuff, along with Russ Whats-his-name, with the Z-boys hot on their tail. I had a GT (GrenTech) plastic board with Power Paw urethane wheels (loose bearings).<br /><br />My parents told me we were moving to California later that year and that they were tak

    File 3: 1622_9.txt
    Lines in file: 1
    First 5 lines (or fewer):
    First i gotta say that this film is way less pretentious than The Da Vinci Code. sure, you have the religion vs science problem but it doesn't try to make a big statement about it. its basically an action thriller that moves from one scene to another very well. one scene particularly (that involves fire) i found extremely well done. <br /><br />Second, the changes from book to film. although when i was following the development of the film i complained about the change of some characters and the complete removal of others, i gotta admit i was wrong. it was refre

    File 4: 3484_10.txt
    Lines in file: 1
    First 5 lines (or fewer):
    This movie felt so real. I actually felt all of the emotions portrayed here during my life at various times + that of both Rory and Michael. I have Duchene's Muscular Dystrophy like Rory so what you see here is exactly what I've actually felt myself. Some won't believe there ARE disabled people like Rory, full of anger and rebellion. I know they exist because I'm one of them.<br /><br />The story is great. For a drama, character-driven movie, the story moves fast. I was never bored, maybe partly because I was seeing stuff that is close to my heart. But I think m

    File 5: 2830_9.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I would suggest that Only the Valiant is one of the most original and intriguing and in some ways weird movies that Peck ever did; daring , surprising and one of his few best westerns (--no, no, of course, not a western really, but a military chronicle, which sometimes is better--). It's quite low budget, but, oh, very original and striking. It's one of those treats a true buff sometimes gets; movies that no one yet told you they exist. You say that sounds intriguing, or interesting and it surpasses your expectations.<br /><br />All in all, the script shows a
  Sentiment: neg

    File 1: 714_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I find Herzog's documentary work to be very uneven. Fata Morgana, a companion piece of sorts to Lessons of Darkness, lacks not only the harrowing spectacle but mostly the discerning eye of an author. It is by comparison amateur looking, aimless pans left and right across the desert the kind of which you would expect from any German tourist equipped with a handycam, the camera left running from the window of a car picking up all kinds of meaningless images, wire fences, derelict buildings and patches of dirt going through the lens in haphazard order, intercut wit

    File 2: 7086_4.txt
    Lines in file: 1
    First 5 lines (or fewer):
    An intelligent summation of Cold War era mutually assured destruction policy, up until the conclusion: it's a gasser! All Russians and Chinese are obliterated from the face of the Earth! Saw this one at the 27th Annual CWRU Science Fiction Marathon, January 2002.

    File 3: 596_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Evil Behind You, was created for a specific purpose in mind, to shove the writer/directors personal views on who either gets to walk on water or who gets to dance with the devil. Sadly it would seem that the creators were so focused on making their point that they took it's power away completely by force feeding their point to the viewer.The way its message is presented Almost reminds me of the stories I've heard of the Spanish inquisition! From one real Christian to another, Avoid this like the plague, fear tactics never work when trying to send this kind of me

    File 4: 7711_1.txt
    Lines in file: 1
    First 5 lines (or fewer):
    I will keep it to bullet points so here goes: 1. Very badly scripted. 2. Tries to be like Resident Evil. 3. Zombies slow and docile one minute the next minute Raging lunatics. 4. Never saw blood clean up so easily! 5. Special effects not as good as the original "day of the living dead". 6. Acting not as good as the "Bold and the beautiful". 7. It looks like it was written in 1 week and made the next week.<br /><br />Basicaly Med team plus Special Forces go into a Zombie infested university to find the first Zombie and extract a serum to cure the plague. All die

    File 5: 4346_3.txt
    Lines in file: 1
    First 5 lines (or fewer):
    Not "confusing" in the sense that, "Gee, this movie is really complex, and thus hard to follow!" But confusing in the sense that, "Gee, this movie really has no idea what it's doing!"<br /><br />DREAM OF A WARRIOR is a Hong Kong/South Korean collaboration, but it's all utter nonsense. A movie about parallel universes mixed in with time travel mixed in with love story mixed in with silliness.<br /><br />The film has the type of concept that boggles the mind. Again, not boggles the mind because it's so great and complex, but boggles in the sense that it's so ridic
```

```
# Prepare directories
batch_size = 32
base_dir = pathlib.Path("aclImdb")
val_dir = base_dir / "val"
train_dir = base_dir / "train"
for category in ("neg", "pos"):
    os.makedirs(val_dir / category, exist_ok=True)  # Allow existing directories
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples = int(0.2 * len(files))
    val_files = files[-num_val_samples:]
    for fname in val_files:
        # Only move files if they don't already exist at the destination
        if not os.path.exists(val_dir / category / fname):
            shutil.move(train_dir / category / fname, val_dir / category / fname)

# Load datasets
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train", batch_size=batch_size
)
val_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/val", batch_size=batch_size
)
test_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=batch_size
)
text_only_train_ds = train_ds.map(lambda x, y: x)
```

```
Found 20000 files belonging to 2 classes.
Found 5000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

```
# Set parameters
max_length = 150
max_tokens = 10000
text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
    output_sequence_length=max_length,
)
text_vectorization.adapt(text_only_train_ds)
```

```
# Tokenized datasets
int_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4).take(100)  # Restrict training samples to 100
int_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4).take(10000)  # Restrict validation samples to 10,000
int_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
```

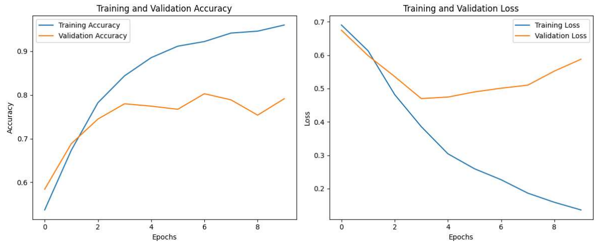
```
# Model with an Embedding Layer
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens, output_dim=128)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
embedding_model = keras.Model(inputs, outputs)

embedding_model.compile(optimizer="rmsprop",
                        loss="binary_crossentropy",
                        metrics=["accuracy"])
embedding_model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("embedding_model.keras", save_best_only=True)
```

```
]
history_embedded = embedding_model.fit(
    int_train_ds, validation_data=int_val_ds, epochs=10, callbacks=callbacks
)

import matplotlib.pyplot as plt

# Extract the history of metrics
history = history_embedded.history

# Plot training and validation accuracy
plt.figure(figsize=(12, 5))

# Subplot for accuracy
plt.subplot(1, 2, 1)
plt.plot(history['accuracy'], label='Training Accuracy')
plt.plot(history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Subplot for loss
plt.subplot(1, 2, 2)
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show the graph
plt.tight_layout()
plt.show()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, None)	0
embedding (Embedding)	(None, None, 128)	1,280,000
bidirectional (Bidirectional)	(None, 64)	41,216
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 1)	65

```
 Total params: 1,321,281 (5.04 MB)
 Trainable params: 1,321,281 (5.04 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 30ms/step - accuracy: 0.5148 - loss: 0.6924 - val_accuracy: 0.5842 - val_loss: 0.6749
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.6383 - loss: 0.6428 - val_accuracy: 0.6884 - val_loss: 0.5987
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 36ms/step - accuracy: 0.7611 - loss: 0.5135 - val_accuracy: 0.7448 - val_loss: 0.5359
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.8254 - loss: 0.4082 - val_accuracy: 0.7798 - val_loss: 0.4699
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.8891 - loss: 0.3126 - val_accuracy: 0.7744 - val_loss: 0.4744
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9150 - loss: 0.2721 - val_accuracy: 0.7672 - val_loss: 0.4898
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.9226 - loss: 0.2324 - val_accuracy: 0.8028 - val_loss: 0.5009
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.9481 - loss: 0.1754 - val_accuracy: 0.7888 - val_loss: 0.5101
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9516 - loss: 0.1461 - val_accuracy: 0.7538 - val_loss: 0.5523
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9731 - loss: 0.1059 - val_accuracy: 0.7914 - val_loss: 0.5879
```




```
#Model with Pretrained Word Embeddings

# Download GloVe embeddings
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip

# Prepare GloVe embedding matrix
embedding_dim = 100
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))

embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

# Pretrained embedding model
embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)

inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pretrained_model = keras.Model(inputs, outputs)

pretrained_model.compile(optimizer="rmsprop",
                         loss="binary_crossentropy",
                         metrics=["accuracy"])
pretrained_model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("pretrained_model.keras", save_best_only=True)
]
history_pretrained = pretrained_model.fit(
    int_train_ds, validation_data=int_val_ds, epochs=10, callbacks=callbacks
)
import matplotlib.pyplot as plt

# Extract the history of metrics
history = history_pretrained.history

# Create a figure for the plots
plt.figure(figsize=(12, 5))

# Subplot for accuracy
plt.subplot(1, 2, 1)
plt.plot(history['accuracy'], label='Training Accuracy')
plt.plot(history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

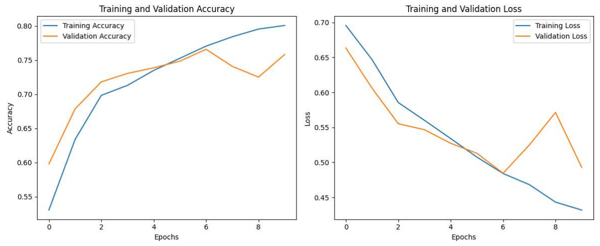
```
# Subplot for loss
plt.subplot(1, 2, 2)
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show the graph
plt.tight_layout()
plt.show()
```

```
--2024-11-26 15:53:16--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2024-11-26 15:53:16--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2024-11-26 15:53:16--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  5.08MB/s    in 2m 40s

2024-11-26 15:55:56 (5.15 MB/s) - 'glove.6B.zip' saved [862182613/862182613]
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, None)	0	-
embedding_1 (Embedding)	(None, None, 100)	1,000,000	input_layer_1[0][0]
not_equal (NotEqual)	(None, None)	0	input_layer_1[0][0]
bidirectional_1 (Bidirectional)	(None, 64)	34,048	embedding_1[0][0], not_equal[0][0]
dropout_1 (Dropout)	(None, 64)	0	bidirectional_1[0][0]
dense_1 (Dense)	(None, 1)	65	dropout_1[0][0]

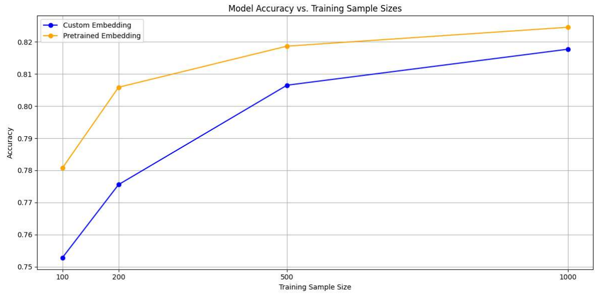
```
Total params: 1,034,113 (3.94 MB)
Trainable params: 34,113 (133.25 KB)
Non-trainable params: 1,000,000 (3.81 MB)
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 51ms/step - accuracy: 0.5059 - loss: 0.7176 - val_accuracy: 0.5978 - val_loss: 0.6635
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 76ms/step - accuracy: 0.6070 - loss: 0.6601 - val_accuracy: 0.6786 - val_loss: 0.6060
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 47ms/step - accuracy: 0.6951 - loss: 0.5955 - val_accuracy: 0.7182 - val_loss: 0.5553
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.7176 - loss: 0.5542 - val_accuracy: 0.7306 - val_loss: 0.5467
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 7s 72ms/step - accuracy: 0.7295 - loss: 0.5368 - val_accuracy: 0.7388 - val_loss: 0.5272
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 63ms/step - accuracy: 0.7504 - loss: 0.5064 - val_accuracy: 0.7486 - val_loss: 0.5130
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - accuracy: 0.7687 - loss: 0.4855 - val_accuracy: 0.7660 - val_loss: 0.4845
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.7788 - loss: 0.4695 - val_accuracy: 0.7410 - val_loss: 0.5247
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - accuracy: 0.7914 - loss: 0.4474 - val_accuracy: 0.7252 - val_loss: 0.5713
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.8046 - loss: 0.4293 - val_accuracy: 0.7584 - val_loss: 0.4929
```




```
import numpy as np
import matplotlib.pyplot as plt
import time

# Data
sample_sizes = [100, 200, 500, 1000]
embedding_accuracies = []
pretrained_accuracies = []

# Initialize the plot
plt.figure(figsize=(12, 6))
plt.title('Model Accuracy vs. Training Sample Sizes')
plt.xlabel('Training Sample Size')
plt.ylabel('Accuracy')
plt.grid(True)

# Iterate over sample sizes
for i, size in enumerate(sample_sizes):
    print(f"\n### Training with {size} samples ###\n")

    # Update training dataset with the specific size
    small_train_ds = train_ds.map(
        lambda x, y: (text_vectorization(x), y)).take(size)

    # Train custom embedding model
    print(f"Training Custom Embedding Model with {size} samples:")
    embedding_model.fit(
        small_train_ds,
        validation_data=int_val_ds,
        epochs=10,
        verbose=1  # Displays training progress for each epoch
    )
    embedding_acc = embedding_model.evaluate(int_test_ds, verbose=1)[1]
    embedding_accuracies.append(embedding_acc)
    print(f"Custom Embedding Model Accuracy: {embedding_acc:.4f}\n")

    # Train pretrained embedding model
    print(f"Training Pretrained Embedding Model with {size} samples:")
    pretrained_model.fit(
        small_train_ds,
        validation_data=int_val_ds,
        epochs=10,
        verbose=1  # Displays training progress for each epoch
    )
    pretrained_acc = pretrained_model.evaluate(int_test_ds, verbose=1)[1]
    pretrained_accuracies.append(pretrained_acc)
    print(f"Pretrained Embedding Model Accuracy: {pretrained_acc:.4f}\n")

    # Update the plot dynamically after each iteration
    if i == len(sample_sizes) - 1:  # Once all iterations are done, plot the final line graph
        plt.plot(sample_sizes, embedding_accuracies, marker='o', label='Custom Embedding', color='blue')
        plt.plot(sample_sizes, pretrained_accuracies, marker='o', label='Pretrained Embedding', color='orange')

# Final plot styling
plt.title('Model Accuracy vs. Training Sample Sizes')
plt.xlabel('Training Sample Size')
plt.ylabel('Accuracy')
plt.xticks(sample_sizes)  # Set x-ticks to sample sizes
plt.grid(True)
plt.legend()

# Show the final plot
plt.tight_layout()
plt.show()
```

```
### Training with 100 samples ###

Training Custom Embedding Model with 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.9829 - loss: 0.0654 - val_accuracy: 0.7376 - val_loss: 0.8625
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9788 - loss: 0.0793 - val_accuracy: 0.7842 - val_loss: 0.6748
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9836 - loss: 0.0693 - val_accuracy: 0.7692 - val_loss: 0.7484
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9917 - loss: 0.0412 - val_accuracy: 0.7732 - val_loss: 0.6976
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.9934 - loss: 0.0242 - val_accuracy: 0.7460 - val_loss: 0.8597
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.9870 - loss: 0.0482 - val_accuracy: 0.6838 - val_loss: 1.3979
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.9918 - loss: 0.0390 - val_accuracy: 0.7490 - val_loss: 0.7969
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9948 - loss: 0.0204 - val_accuracy: 0.7740 - val_loss: 0.8716
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9956 - loss: 0.0177 - val_accuracy: 0.7316 - val_loss: 1.2452
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9985 - loss: 0.0119 - val_accuracy: 0.7614 - val_loss: 0.8887
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.7528 - loss: 0.9349
Custom Embedding Model Accuracy: 0.7528

Training Pretrained Embedding Model with 100 samples:
Epoch 1/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.8178 - loss: 0.4021 - val_accuracy: 0.7700 - val_loss: 0.4729
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 27ms/step - accuracy: 0.8281 - loss: 0.3918 - val_accuracy: 0.7750 - val_loss: 0.4723
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.8402 - loss: 0.3746 - val_accuracy: 0.7700 - val_loss: 0.4831
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 46ms/step - accuracy: 0.8371 - loss: 0.3715 - val_accuracy: 0.7796 - val_loss: 0.4706
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 31ms/step - accuracy: 0.8475 - loss: 0.3406 - val_accuracy: 0.7660 - val_loss: 0.5105
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.8518 - loss: 0.3405 - val_accuracy: 0.7628 - val_loss: 0.5107
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.8625 - loss: 0.3334 - val_accuracy: 0.7770 - val_loss: 0.4813
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.8768 - loss: 0.3042 - val_accuracy: 0.7792 - val_loss: 0.4873
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.8827 - loss: 0.2863 - val_accuracy: 0.7640 - val_loss: 0.5608
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 6s 51ms/step - accuracy: 0.8878 - loss: 0.2756 - val_accuracy: 0.7788 - val_loss: 0.5052
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.7823 - loss: 0.4930
Pretrained Embedding Model Accuracy: 0.7807


### Training with 200 samples ###

Training Custom Embedding Model with 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9599 - loss: 0.1093 - val_accuracy: 0.7948 - val_loss: 0.4813
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 6s 31ms/step - accuracy: 0.9703 - loss: 0.0873 - val_accuracy: 0.7820 - val_loss: 0.5157
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9774 - loss: 0.0739 - val_accuracy: 0.8042 - val_loss: 0.4892
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9800 - loss: 0.0727 - val_accuracy: 0.8092 - val_loss: 0.5358
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 6s 28ms/step - accuracy: 0.9860 - loss: 0.0436 - val_accuracy: 0.7894 - val_loss: 0.8231
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 9s 23ms/step - accuracy: 0.9865 - loss: 0.0412 - val_accuracy: 0.7978 - val_loss: 0.6332
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9902 - loss: 0.0372 - val_accuracy: 0.7766 - val_loss: 0.7011
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 5s 24ms/step - accuracy: 0.9942 - loss: 0.0220 - val_accuracy: 0.7938 - val_loss: 0.7965
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 7s 34ms/step - accuracy: 0.9948 - loss: 0.0177 - val_accuracy: 0.8070 - val_loss: 0.9817
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 8s 20ms/step - accuracy: 0.9973 - loss: 0.0127 - val_accuracy: 0.7922 - val_loss: 1.1377
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.7748 - loss: 1.1705
Custom Embedding Model Accuracy: 0.7756

Training Pretrained Embedding Model with 200 samples:
Epoch 1/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 6s 29ms/step - accuracy: 0.8764 - loss: 0.2959 - val_accuracy: 0.7826 - val_loss: 0.4512
Epoch 2/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.8742 - loss: 0.3123 - val_accuracy: 0.7322 - val_loss: 0.5657
Epoch 3/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.8721 - loss: 0.2983 - val_accuracy: 0.8014 - val_loss: 0.4347
Epoch 4/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 7s 33ms/step - accuracy: 0.8819 - loss: 0.2787 - val_accuracy: 0.7860 - val_loss: 0.4621
Epoch 5/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 8s 21ms/step - accuracy: 0.8943 - loss: 0.2689 - val_accuracy: 0.7900 - val_loss: 0.4818
Epoch 6/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.8946 - loss: 0.2593 - val_accuracy: 0.7820 - val_loss: 0.4782
Epoch 7/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 6s 28ms/step - accuracy: 0.9033 - loss: 0.2376 - val_accuracy: 0.7920 - val_loss: 0.4711
Epoch 8/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 7s 35ms/step - accuracy: 0.9060 - loss: 0.2300 - val_accuracy: 0.7878 - val_loss: 0.4566
Epoch 9/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9117 - loss: 0.2193 - val_accuracy: 0.7970 - val_loss: 0.4805
Epoch 10/10
200/200 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - accuracy: 0.9240 - loss: 0.2015 - val_accuracy: 0.7988 - val_loss: 0.5067
782/782 ━━━━━━━━━━━━━━━━━━━━ 10s 13ms/step - accuracy: 0.8056 - loss: 0.4963
Pretrained Embedding Model Accuracy: 0.8059


### Training with 500 samples ###

Training Custom Embedding Model with 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 9s 18ms/step - accuracy: 0.9480 - loss: 0.1358 - val_accuracy: 0.8244 - val_loss: 0.4081
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - accuracy: 0.9532 - loss: 0.1326 - val_accuracy: 0.8258 - val_loss: 0.4167
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.9648 - loss: 0.1043 - val_accuracy: 0.8278 - val_loss: 0.4270
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9732 - loss: 0.0809 - val_accuracy: 0.8184 - val_loss: 0.4695
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 11s 22ms/step - accuracy: 0.9820 - loss: 0.0579 - val_accuracy: 0.8228 - val_loss: 0.5724
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 21s 22ms/step - accuracy: 0.9849 - loss: 0.0437 - val_accuracy: 0.8202 - val_loss: 0.6989
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 20s 22ms/step - accuracy: 0.9878 - loss: 0.0323 - val_accuracy: 0.8190 - val_loss: 0.7558
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.9936 - loss: 0.0214 - val_accuracy: 0.8140 - val_loss: 0.7240
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9925 - loss: 0.0216 - val_accuracy: 0.8222 - val_loss: 0.8802
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 11s 22ms/step - accuracy: 0.9952 - loss: 0.0159 - val_accuracy: 0.8186 - val_loss: 0.9529
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.8038 - loss: 1.0619
Custom Embedding Model Accuracy: 0.8065

Training Pretrained Embedding Model with 500 samples:
Epoch 1/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 12s 23ms/step - accuracy: 0.8857 - loss: 0.2691 - val_accuracy: 0.8114 - val_loss: 0.4183
Epoch 2/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.8826 - loss: 0.2855 - val_accuracy: 0.8234 - val_loss: 0.3936
Epoch 3/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 12s 23ms/step - accuracy: 0.8843 - loss: 0.2770 - val_accuracy: 0.8228 - val_loss: 0.3919
Epoch 4/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.8920 - loss: 0.2595 - val_accuracy: 0.8120 - val_loss: 0.4090
Epoch 5/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - accuracy: 0.8987 - loss: 0.2496 - val_accuracy: 0.8302 - val_loss: 0.3962
Epoch 6/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - accuracy: 0.9021 - loss: 0.2386 - val_accuracy: 0.8158 - val_loss: 0.4208
Epoch 7/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 12s 22ms/step - accuracy: 0.9089 - loss: 0.2260 - val_accuracy: 0.8146 - val_loss: 0.4202
Epoch 8/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 21ms/step - accuracy: 0.9171 - loss: 0.2101 - val_accuracy: 0.8306 - val_loss: 0.4063
Epoch 9/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 11s 23ms/step - accuracy: 0.9184 - loss: 0.2008 - val_accuracy: 0.8216 - val_loss: 0.4178
Epoch 10/10
500/500 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 0.9268 - loss: 0.1883 - val_accuracy: 0.8210 - val_loss: 0.4480
782/782 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - accuracy: 0.8167 - loss: 0.4520
Pretrained Embedding Model Accuracy: 0.8187


### Training with 1000 samples ###

Training Custom Embedding Model with 1000 samples:
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - accuracy: 0.9927 - loss: 0.0237 - val_accuracy: 0.8270 - val_loss: 0.4541
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9908 - loss: 0.0300 - val_accuracy: 0.8270 - val_loss: 0.5086
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 22ms/step - accuracy: 0.9935 - loss: 0.0208 - val_accuracy: 0.8172 - val_loss: 0.6668
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.9951 - loss: 0.0165 - val_accuracy: 0.8190 - val_loss: 0.7536
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.9967 - loss: 0.0119 - val_accuracy: 0.8284 - val_loss: 0.8521
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 24ms/step - accuracy: 0.9971 - loss: 0.0114 - val_accuracy: 0.8236 - val_loss: 1.0514
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 18ms/step - accuracy: 0.9970 - loss: 0.0088 - val_accuracy: 0.8238 - val_loss: 1.0885
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - accuracy: 0.9981 - loss: 0.0055 - val_accuracy: 0.8282 - val_loss: 1.1604
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 20ms/step - accuracy: 0.9984 - loss: 0.0048 - val_accuracy: 0.8248 - val_loss: 1.2800
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 23ms/step - accuracy: 0.9991 - loss: 0.0036 - val_accuracy: 0.8260 - val_loss: 1.3064
782/782 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.8155 - loss: 1.3611
Custom Embedding Model Accuracy: 0.8177

Training Pretrained Embedding Model with 1000 samples:
Epoch 1/10
```

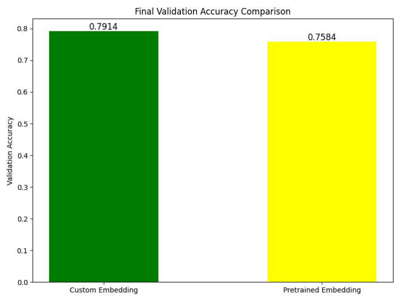
```
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 24ms/step - accuracy: 0.9204 - loss: 0.2030 - val_accuracy: 0.8328 - val_loss: 0.3877
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 17ms/step - accuracy: 0.9267 - loss: 0.1858 - val_accuracy: 0.8334 - val_loss: 0.3858
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 14s 23ms/step - accuracy: 0.9337 - loss: 0.1748 - val_accuracy: 0.8366 - val_loss: 0.3921
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.9347 - loss: 0.1693 - val_accuracy: 0.8302 - val_loss: 0.4066
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 12s 19ms/step - accuracy: 0.9407 - loss: 0.1555 - val_accuracy: 0.8286 - val_loss: 0.4259
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 21s 20ms/step - accuracy: 0.9439 - loss: 0.1467 - val_accuracy: 0.8222 - val_loss: 0.4430
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 13s 21ms/step - accuracy: 0.9499 - loss: 0.1387 - val_accuracy: 0.8264 - val_loss: 0.4309
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 15s 24ms/step - accuracy: 0.9501 - loss: 0.1338 - val_accuracy: 0.8270 - val_loss: 0.4668
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 11s 18ms/step - accuracy: 0.9517 - loss: 0.1229 - val_accuracy: 0.8228 - val_loss: 0.4887
782/782 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.8252 - loss: 0.4779
Pretrained Embedding Model Accuracy: 0.8246
```




```python
import pandas as pd

# Collect results for custom embedding model
custom_embedding_results = {
    "Sample Size": sample_sizes,
    "Custom Embedding Accuracy": embedding_accuracies,
}

# Collect results for pretrained embedding model
pretrained_embedding_results = {
    "Sample Size": sample_sizes,
    "Pretrained Embedding Accuracy": pretrained_accuracies,
}

# Combine into a single DataFrame
summary_df = pd.DataFrame({
    "Sample Size": sample_sizes,
    "Custom Embedding Accuracy": embedding_accuracies,
    "Pretrained Embedding Accuracy": pretrained_accuracies
})

# Display the table
print("Summary of Results:")
print(summary_df)
```

```
Summary of Results:
   Sample Size  Custom Embedding Accuracy  Pretrained Embedding Accuracy
0          100                    0.75280                        0.78072
1          200                    0.77556                        0.80588
2          500                    0.80652                        0.81868
3         1000                    0.81772                        0.82456
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Data
models = ['Custom Embedding', 'Pretrained Embedding']
final_val_accuracies = [
    history_embedded.history['val_accuracy'][-1],  # Final validation accuracy for custom embedding
    history_pretrained.history['val_accuracy'][-1]  # Final validation accuracy for pretrained embedding
]

# Bar plot
plt.figure(figsize=(8, 6))
plt.bar(models, final_val_accuracies, color=['green', 'yellow'], width=0.5)

# Add labels and title
plt.ylabel('Validation Accuracy')
plt.title('Final Validation Accuracy Comparison')

# Display final values on top of bars
for i, v in enumerate(final_val_accuracies):
    plt.text(i, v + 0.005, f'{v:.4f}', ha='center', fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()
```




```python
import numpy as np
import matplotlib.pyplot as plt

# Data
sample_sizes = [100, 200, 500, 1000]
embedding_accuracies = [0.75280, 0.77556, 0.80652, 0.81772]  # Replace with actual final values
pretrained_accuracies = [0.78072, 0.80588, 0.81868, 0.82456]  # Replace with actual final values

# Bar width and positions
bar_width = 0.15
x_indices = np.arange(len(sample_sizes))  # Base positions for bars

# Plot bars
plt.figure(figsize=(11, 6))
plt.bar(x_indices - bar_width / 2, embedding_accuracies,
        width=bar_width, label='Custom Embedding', color='skyblue')
plt.bar(x_indices + bar_width / 2, pretrained_accuracies,
        width=bar_width, label='Pretrained Embedding', color='yellow')

# Overlay lines for comparison
plt.plot(x_indices - bar_width / 2, embedding_accuracies,
         marker='o', color='orange', linestyle='--', label='Custom Embedding (Line)')
plt.plot(x_indices + bar_width / 2, pretrained_accuracies,
         marker='o', color='grey', linestyle='--', label='Pretrained Embedding (Line)')
```

```
# Add labels and title
plt.xlabel('Training Sample Size', fontsize=12)
plt.ylabel('Final Test Accuracy', fontsize=12)
plt.title('Final Test Accuracy vs. Training Sample Size', fontsize=14)
plt.xticks(x_indices, sample_sizes)  # Use sample sizes as x-tick labels
plt.legend()

# Annotate bars with final accuracy values
for i in range(len(sample_sizes)):
    plt.text(x_indices[i] - bar_width / 2, embedding_accuracies[i] + 0.002,
             f'{embedding_accuracies[i]:.4f}', ha='center', fontsize=10)
    plt.text(x_indices[i] + bar_width / 2, pretrained_accuracies[i] + 0.002,
             f'{pretrained_accuracies[i]:.4f}', ha='center', fontsize=10)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```